Security Audit Report for Deltatrade on Solana

Date: September 10, 2024 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Deltatrade
Target	Deltatrade on Solana

Version History

Version	Date	Description
1.0	September 10, 2024	First release

Signature

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Rust
Approach	Semi-automatic and manual verification

The audit focuses on the Deltatrade on Solana of Deltatrade [1], which allows users to automatically buy or sell assets with specific prices by creating grids.

The audit includes only files within the `contract-solana/programs` folder of the repository. Other files are excluded. External dependencies, including the Solana development framework `Anchor` [2], are considered reliable for functionality and security and are not within the audit's scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report.

Project	Version	Commit Hash
Deltatrade on Solana	`Version 1`	`62c4a222b88de1a6e21c977a33633f0ade91ce31`
	`Version 2`	`ba7e79d6bea2429c78daf3f137cab80cd9eeda9e`

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly

[1] `https://github.com/DeltaBotDev/contract-solana`

[2] `https://www.anchor-lang.com/`

specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

- Reentrancy
- DoS
- Access control
- Data handling and data flow
- Exception handling
- Untrusted external call and control flow
- Initialization consistency
- Events operation
- Error-prone randomness
- Improper use of the proxy system

1.3.2 DeFi Security

- Semantic consistency
- Functionality consistency
- Permission management
- Business logic
- Token operation
- Emergency mechanism
- Oracle security
- Whitelist and blacklist
- Economic impact
- Batch transfer

1.3.3 NFT Security

- Duplicated item

* Verification of the token receiver
* Off-chain metadata security

1.3.4 Additional Recommendation

* Gas optimization
* Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [3] and Common Weakness Enumeration [4]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ **Likelihood**	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

[3] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[4] https://cwe.mitre.org/

Chapter 2 Findings

In total, we found **five** potential security issue. Besides, we have **eight** recommendations and **one** note.

- High Risk: 5
- Recommendation: 8
- Note: 1

ID	Severity	Description	Category	Status
1	High	Potential overflow due to incorrect use of data type	Software Security	Fixed
2	High	Lack of check on the types of funds corresponding to `grid_bot`	DeFi Security	Fixed
3	High	Incorrect amount of revenue token withdrawn in function `close_bot()`	DeFi Security	Fixed
4	High	Incorrect logic implemented in function `round_up()`	DeFi Security	Fixed
5	High	Incorrect token account and corresponding authority used in function `close_bot()`	DeFi Security	Fixed
6	-	Lack of check for the length of `maker_users`	Recommendation	Fixed
7	-	Redundant status configured in `grid_bot`	Recommendation	Fixed
8	-	Lack of check for the sum of `grid_sell_count` and `grid_buy_count`	Recommendation	Fixed
9	-	Saving bumps when initializing PDA accounts for later use	Recommendation	Confirmed
10	-	Lack of access control during the initialization	Recommendation	Fixed
11	-	Redundant accounts required in function `register_pair()`	Recommendation	Fixed
12	-	Lack of check in function `set_owner()`	Recommendation	Confirmed
13	-	Lack of duplication check in function `set_maker_user()`	Recommendation	Confirmed
14	-	Potential centralization risks	Note	-

The details are provided in the following sections.

2.1 Software Security

2.1.1 Potential overflow due to incorrect use of data type

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the function `internal_check_order_match()`, the expression `take_orders_param.amount_sell * maker_order.amount_sell >= take_orders_param.amount_buy`

`* maker_order.amount_buy` uses `u64` data types, which poses a risk of overflow. The same issue is present in the function `internal_calculate_matching()`. Meanwhile, in the function `internal_get_first_forward_order()`, the expression `(grid_rate_denominator_128 + grid_bot.grid_rate as u128).pow(level as u32)` uses the `u128` data type, capable of representing values up to approximately `1e39`. Given `grid_rate_denominator_128` is set to `1e4`, computing `grid_rate_denominator_128.pow(level)` can cause an overflow with relatively small values of level.

```rust
pub fn internal_check_order_match(maker_order: &Order, take_orders_param: &TakeOrdersParam, pair
    : &Pair) -> Result<()> {
    let (maker_sell_token, maker_buy_token) = GridProcess::internal_get_order_token_sell_and_buy
        (maker_order, pair);
    require!(maker_buy_token == take_orders_param.token_sell, Errors::InvalidOrderToken);
    require!(maker_sell_token == take_orders_param.token_buy, Errors::InvalidOrderToken);
    require!(take_orders_param.token_sell != take_orders_param.token_buy, Errors::
        InvalidOrderToken);
    // taker price and maker price match
    // msg!("order check taker sell:{}, taker buy:{}, maker buy:{}, maker sell:{}",
        take_orders_param.amount_sell.to_string(), take_orders_param.amount_buy.to_string(),
        maker_order.amount_buy.to_string(), maker_order.amount_sell.to_string());
    require!(take_orders_param.amount_sell * maker_order.amount_sell >= take_orders_param.
        amount_buy * maker_order.amount_buy, Errors::OrderPriceNotMatch);
    return Ok(())
}
```

Listing 2.1: src/process/grid/order_check.rs

```rust
pub fn internal_calculate_matching(maker_order: &Order, taker_order: &Order, took_sell: u64,
    took_buy: u64) -> (u64, u64, u64, Order) {
    // calculate marker max amount
    let max_fill_sell;
    let max_fill_buy;
    if maker_order.fill_buy_or_sell {
        max_fill_buy = maker_order.amount_buy - maker_order.filled;
        max_fill_sell = maker_order.amount_sell * max_fill_buy / maker_order.amount_buy;
    } else {
        max_fill_sell = maker_order.amount_sell - maker_order.filled;
        max_fill_buy = common::round_up(BigDecimal::from(maker_order.amount_buy) * (BigDecimal::
            from(max_fill_sell)) / (BigDecimal::from(maker_order.amount_sell)), 0).to_u64().
            unwrap();
    }
    // calculate matching amount
    let taker_sell;
    let taker_buy;
    if taker_order.fill_buy_or_sell {
        let max_taker_buy = taker_order.amount_buy - took_buy;
        if max_taker_buy >= max_fill_sell {
            // taker all maker
            taker_buy = max_fill_sell;
            taker_sell = max_fill_buy;
        } else {
            taker_buy = max_taker_buy;
            taker_sell = common::round_up(BigDecimal::from(max_fill_buy) * (BigDecimal::from(
                taker_buy)) / (BigDecimal::from(max_fill_sell)), 0).to_u64().unwrap();
        }
    } else {
        let max_taker_sell = taker_order.amount_sell - took_sell;
        if max_taker_sell >= max_fill_buy {
            // taker all maker
            taker_buy = max_fill_sell;
            taker_sell = max_fill_buy;
        } else {
            taker_sell = max_taker_sell;
            taker_buy = (max_fill_sell as u128 * (taker_sell as u128) / (max_fill_buy as u128))
                as u64;
        }
    }
    let current_filled= if maker_order.fill_buy_or_sell {
        taker_sell.clone()
    } else {
        taker_buy.clone()
    };
    let mut made_order = maker_order.clone();
    made_order.amount_sell = taker_buy.clone();
    made_order.amount_buy = taker_sell.clone();
    made_order.filled = 0;


    return (taker_sell, taker_buy, current_filled, made_order);
}
```

Listing 2.2: src/process/grid/order_calculate.rs

```
pub fn internal_get_first_forward_order(grid_bot: &GridBot, level: u64) -> Order {
    let mut order = Order{
        token_sell_is_base: true,
        amount_sell: 0,
        amount_buy: 0,
        fill_buy_or_sell: false,
        filled: 0,
    };
    let grid_rate_denominator_128 = GRID_RATE_DENOMINATOR as u128;
    if grid_bot.grid_buy_count > (level as u16) {
        // buy grid
        order.token_sell_is_base = false;
        order.fill_buy_or_sell = grid_bot.fill_base_or_quote;
        if grid_bot.fill_base_or_quote {
            // fixed base
            order.amount_buy = grid_bot.first_base_amount;
            order.amount_sell = if grid_bot.grid_type == GRID_TYPE_EQ_OFFSET {
                // arithmetic grid
                grid_bot.first_quote_amount + grid_bot.grid_offset * level
            } else {
                // proportional grid
                ((grid_bot.first_quote_amount as u128) * (grid_rate_denominator_128 + grid_bot.
                    grid_rate as u128).pow(level as u32) / grid_rate_denominator_128.pow(level as
                     u32)) as u64
            };
        } else {
            // fixed quote
            order.amount_sell = grid_bot.first_quote_amount.clone();
            order.amount_buy = if grid_bot.grid_type == GRID_TYPE_EQ_OFFSET {
                // arithmetic grid
                grid_bot.first_base_amount - grid_bot.grid_offset * level
            } else {
                // proportional grid
                ((grid_bot.first_base_amount as u128) * grid_rate_denominator_128.pow(level as
                    u32) / ((grid_rate_denominator_128 + grid_bot.grid_rate as u128).pow(level as
                     u32))) as u64
            };
        }
    } else {
        // sell grid
        order.token_sell_is_base = true;
        order.fill_buy_or_sell = !grid_bot.fill_base_or_quote;
        let coefficient = (grid_bot.grid_buy_count + grid_bot.grid_sell_count - 1 - level as u16
            ) as u64;
        if grid_bot.fill_base_or_quote {
            // fixed base
            order.amount_sell = grid_bot.last_base_amount;
            order.amount_buy = if grid_bot.grid_type == GRID_TYPE_EQ_OFFSET {
                grid_bot.last_quote_amount - grid_bot.grid_offset * coefficient
            } else {
                ((grid_bot.last_quote_amount as u128) * grid_rate_denominator_128.pow(coefficient
                     as u32) / ((grid_rate_denominator_128 + grid_bot.grid_rate as u128).pow(
                    coefficient as u32))) as u64
            };
        } else {
            // fixed quote
            order.amount_buy = grid_bot.last_quote_amount;
            order.amount_sell = if grid_bot.grid_type == GRID_TYPE_EQ_OFFSET {
                grid_bot.last_base_amount + grid_bot.grid_offset * coefficient
            } else {
                ((grid_bot.last_base_amount as u128) * (grid_rate_denominator_128 + grid_bot.
                    grid_rate as u128).pow(coefficient as u32) / grid_rate_denominator_128.pow(
                    coefficient as u32)) as u64
            };
        }
    }
    return order;
}
```

Listing 2.3: src/process/grid/order_calculate.rs

Impact Protocol is not functioning properly due to overflow.

Suggestion Use appropriate data types to prevent overflow.

2.2 DeFi Security

2.2.1 Lack of check on the types of funds corresponding to `grid_bot`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In function `close_bot()`, there is no verification to ensure that the `base_mint` and `quote_mint` in the `pair` provided by the user match the `grid_bot`. In this case, the owner of the `grid_bot` can provide a more valuable pair and withdraw the corresponding tokens when closing the `grid_bot` for profit.

The similar issue also occurs in function `claim()`.

```rust
#[derive(Accounts)]
#[instruction(params: CloseBotParam)]
pub struct GridBotClose<'info> {
    pub grid_bot_state: Box<Account<'info, GridBotState>>,
    #[account(seeds = [PDA_SEED_USER_STATE.as_bytes(), grid_bot_state.key().as_ref(), user.key
        ().as_ref()], bump)]
    pub user_state: Box<Account<'info, UserState>>,
    pub base_mint: Box<InterfaceAccount<'info, Mint>>,
    pub quote_mint: Box<InterfaceAccount<'info, Mint>>,
    #[account(seeds = [grid_bot_state.key().as_ref(), base_mint.key().as_ref(), quote_mint.key
        ().as_ref()], bump)]
    pub pair: Box<Account<'info, Pair>>,
    #[account(mut, seeds = [PDA_SEED_USER_GRID_BOT.as_bytes(), grid_bot_state.key().as_ref(),
        user.key().as_ref(), &params.user_state_id.to_be_bytes()], bump)]
    pub grid_bot: Box<Account<'info, GridBot>>,


    /// CHECK:global_balance_base_user
    #[account(seeds = [PDA_SEED_GLOBAL_BALANCE_USER.as_bytes(), grid_bot_state.key().as_ref(),
        base_mint.key().as_ref()], bump)]
    pub global_balance_base_user: UncheckedAccount<'info>,
    #[account(mut, associated_token::mint = base_mint, associated_token::authority =
        global_balance_base_user)]
    pub global_balance_base: Box<InterfaceAccount<'info, TokenAccount>>,


    /// CHECK:global_balance_quote_user
    #[account(seeds = [PDA_SEED_GLOBAL_BALANCE_USER.as_bytes(), grid_bot_state.key().as_ref(),
        quote_mint.key().as_ref()], bump)]
    pub global_balance_quote_user: UncheckedAccount<'info>,
    #[account(mut, associated_token::mint = quote_mint, associated_token::authority =
        global_balance_quote_user)]
    pub global_balance_quote: Box<InterfaceAccount<'info, TokenAccount>>,


    #[account(mut)]
    pub user_base_token_account: Box<InterfaceAccount<'info, TokenAccount>>,
    #[account(mut)]
    pub user_quote_token_account: Box<InterfaceAccount<'info, TokenAccount>>,


    pub token_program: Interface<'info, TokenInterface>,
    pub associated_token_program: Program<'info, AssociatedToken>,
    #[account(mut)]
    pub user: Signer<'info>,
    pub system_program: Program<'info, System>,
}
```

Listing 2.4: src/instructions/grid/close.rs

```
pub fn close_bot(accounts: &mut GridBotClose, close_bot_param: CloseBotParam) -> Result<()> {
    require!(!accounts.grid_bot.closed, Errors::InvalidBotStatus);
    require!(accounts.grid_bot.user == *accounts.user.key, Errors::InvalidUser);

    return GridProcess::internal_close_bot(accounts, close_bot_param);
}

pub fn internal_close_bot(accounts: &mut GridBotClose, close_bot_param: CloseBotParam) ->
     Result<()> {
   // sign closed
   accounts.grid_bot.closed = true;
   // harvest revenue, must fist execute, will split revenue from bot's asset
   let (revenue_token, revenue) = GridProcess::internal_harvest_revenue(accounts.grid_bot.
       as_mut(), accounts.pair.as_ref());

   // withdraw token
   GridProcess::internal_withdraw_asset(&accounts.grid_bot_state.key(), &accounts.base_mint.key
       (), accounts.global_balance_base.to_account_info(), accounts.user_base_token_account.
       to_account_info(),
   accounts.global_balance_base_user.to_account_info(), accounts.token_program.to_account_info
       (), close_bot_param.global_base_bump, accounts.grid_bot.total_base_amount.clone())?;
   GridProcess::internal_withdraw_asset(&accounts.grid_bot_state.key(), &accounts.quote_mint.
       key(), accounts.global_balance_quote.to_account_info(), accounts.
       user_quote_token_account.to_account_info(),
   accounts.global_balance_quote_user.to_account_info(), accounts.token_program.to_account_info
       (), close_bot_param.global_quote_bump, accounts.grid_bot.total_quote_amount.clone())?;
   if revenue_token == accounts.base_mint.key() {
       GridProcess::internal_withdraw_asset(&accounts.grid_bot_state.key(), &accounts.base_mint
           .key(), accounts.global_balance_quote.to_account_info(), accounts.
           user_quote_token_account.to_account_info(),
       accounts.global_balance_quote_user.to_account_info(), accounts.token_program.
           to_account_info(), close_bot_param.global_quote_bump, accounts.grid_bot.
           total_quote_amount.clone())?;
   } else {
       GridProcess::internal_withdraw_asset(&accounts.grid_bot_state.key(), &accounts.
           quote_mint.key(), accounts.global_balance_quote.to_account_info(), accounts.
           user_quote_token_account.to_account_info(),
       accounts.global_balance_quote_user.to_account_info(), accounts.token_program.
           to_account_info(), close_bot_param.global_quote_bump, revenue)?;
   }
   emit!(CloseEvent{
        account_id: accounts.grid_bot.user,
        bot_id: accounts.grid_bot.bot_id.to_string(),
        user_state_id: close_bot_param.user_state_id.to_string(),
        refund: 0,
    });
    Ok(())
}
```

Listing 2.5: src/process/grid/close.rs

```
#[derive(Accounts)]
#[instruction(params: ClaimParam)]
pub struct GridBotClaim<'info> {
    pub grid_bot_state: Box<Account<'info, GridBotState>>,
    pub base_mint: Box<InterfaceAccount<'info, Mint>>,
    pub quote_mint: Box<InterfaceAccount<'info, Mint>>,
    #[account(mut, seeds = [PDA_SEED_USER_GRID_BOT.as_bytes(), grid_bot_state.key().as_ref(),
        user.key().as_ref(), &params.user_state_id.to_be_bytes()], bump)]
    pub grid_bot: Box<Account<'info, GridBot>>,
    #[account(seeds = [grid_bot_state.key().as_ref(), base_mint.key().as_ref(), quote_mint.key
        ().as_ref()], bump)]
    pub pair: Box<Account<'info, Pair>>,


    /// CHECK:global_balance_base_user
    #[account(seeds = [PDA_SEED_GLOBAL_BALANCE_USER.as_bytes(), grid_bot_state.key().as_ref(),
        base_mint.key().as_ref()], bump)]
    pub global_balance_base_user: UncheckedAccount<'info>,
    #[account(mut, associated_token::mint = base_mint, associated_token::authority =
        global_balance_base_user)]
    pub global_balance_base: Box<InterfaceAccount<'info, TokenAccount>>,


    /// CHECK:global_balance_quote_user
    #[account(seeds = [PDA_SEED_GLOBAL_BALANCE_USER.as_bytes(), grid_bot_state.key().as_ref(),
        quote_mint.key().as_ref()], bump)]
    pub global_balance_quote_user: UncheckedAccount<'info>,
    #[account(mut, associated_token::mint = quote_mint, associated_token::authority =
        global_balance_quote_user)]
    pub global_balance_quote: Box<InterfaceAccount<'info, TokenAccount>>,


    #[account(mut)]
    pub user_base_token_account: Box<InterfaceAccount<'info, TokenAccount>>,
    #[account(mut)]
    pub user_quote_token_account: Box<InterfaceAccount<'info, TokenAccount>>,


    pub token_program: Interface<'info, TokenInterface>,
    pub associated_token_program: Program<'info, AssociatedToken>,
    #[account(mut)]
    pub user: Signer<'info>,
    pub system_program: Program<'info, System>,
}
```

Listing 2.6: src/instructions/grid/claim.rs

```
pub fn claim(accounts: &mut GridBotClaim, claim_param: ClaimParam) -> Result<()> {
    require!(accounts.grid_bot.user == accounts.user.key(), Errors::InvalidUser);
    let (revenue_token, revenue) = GridProcess::internal_harvest_revenue(accounts.grid_bot.
        as_mut(), accounts.pair.as_ref());
    let (global_user, global_account, global_bump, user_account) = GridProcess::
        internal_get_user_and_token_account(revenue_token == accounts.global_balance_base.key()
        , accounts.global_balance_base_user.to_account_info(), accounts.global_balance_base.
        to_account_info(), claim_param.global_base_bump,
    accounts.global_balance_quote_user.to_account_info(), accounts.global_balance_quote.
        to_account_info(), claim_param.global_quote_bump,
    accounts.user_base_token_account.to_account_info(), accounts.user_quote_token_account.
        to_account_info());
    // withdraw revenue
    GridProcess::internal_withdraw_asset(
        &accounts.grid_bot_state.key(),&revenue_token, global_account, user_account,
                                        global_user, accounts.token_program.to_account_info(),
                                            global_bump, revenue)?;
    // claim event
    emit!(ClaimEvent{
        claim_user: accounts.user.key(),
        bot_id: accounts.grid_bot.bot_id.to_string(),
        user_state_id: claim_param.user_state_id.to_string(),
        user: accounts.user.key(),
        revenue_token: revenue_token,
        revenue: revenue,
    });
    Ok(())
}
```

Listing 2.7: src/process/grid/claim.rs

Impact The funds of the contract can be drained.

Suggestion Add checks to ensure that the types of tokens in the `grid_bot` correspond to the `pair`.

2.2.2 Incorrect amount of revenue token withdrawn in function `close_bot()`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Users can close their own `grid_bot` by using the function `close_bot()`. In the function `internal_close_bot()`, the procedure involves subtracting the corresponding revenue from the `total_base_amount` or `total_quote_amount` recorded in the `grid_bot`, and then extracting the user's principal and profit from the protocol's token account. However, when the revenue_token is the base_token, the extracted quantity of revenue_token is incorrect. Specifically, the extracted profit should be the revenue rather than the total_quote_amount of the grid_bot.

```rust
pub fn close_bot(ctx: Context<GridBotClose>, close_bot_param: CloseBotParam) -> Result<()> {
    check_context(&ctx)?;
    return GridProcess::close_bot(ctx.accounts, close_bot_param);
}
```

Listing 2.8: lib.rs

```rust
pub fn close_bot(accounts: &mut GridBotClose, close_bot_param: CloseBotParam) -> Result<()> {
    require!(!accounts.grid_bot.closed, Errors::InvalidBotStatus);
    require!(accounts.grid_bot.user == *accounts.user.key, Errors::InvalidUser);

    return GridProcess::internal_close_bot(accounts, close_bot_param);
}
```

Listing 2.9: src/process/grid/close.rs

```rust
pub fn internal_close_bot(accounts: &mut GridBotClose, close_bot_param: CloseBotParam) -> Result
    <()> {
    // sign closed
    accounts.grid_bot.closed = true;
    // harvest revenue, must fist execute, will split revenue from bot's asset
    let (revenue_token, revenue) = GridProcess::internal_harvest_revenue(accounts.grid_bot.
        as_mut(), accounts.pair.as_ref());

    // withdraw token
    GridProcess::internal_withdraw_asset(&accounts.grid_bot_state.key(), &accounts.base_mint.key
        (), accounts.global_balance_base.to_account_info(), accounts.user_base_token_account.
        to_account_info(),
    accounts.global_balance_base_user.to_account_info(), accounts.token_program.to_account_info
        (), close_bot_param.global_base_bump, accounts.grid_bot.total_base_amount.clone())?;
    GridProcess::internal_withdraw_asset(&accounts.grid_bot_state.key(), &accounts.quote_mint.
        key(), accounts.global_balance_quote.to_account_info(), accounts.
        user_quote_token_account.to_account_info(),
    accounts.global_balance_quote_user.to_account_info(), accounts.token_program.to_account_info
        (), close_bot_param.global_quote_bump, accounts.grid_bot.total_quote_amount.clone())?;
    if revenue_token == accounts.base_mint.key() {
        GridProcess::internal_withdraw_asset(&accounts.grid_bot_state.key(), &accounts.base_mint
            .key(), accounts.global_balance_quote.to_account_info(), accounts.
            user_quote_token_account.to_account_info(),
        accounts.global_balance_quote_user.to_account_info(), accounts.token_program.
            to_account_info(), close_bot_param.global_quote_bump, accounts.grid_bot.
            total_quote_amount.clone())?;
    } else {
        GridProcess::internal_withdraw_asset(&accounts.grid_bot_state.key(), &accounts.
            quote_mint.key(), accounts.global_balance_quote.to_account_info(), accounts.
            user_quote_token_account.to_account_info(),
        accounts.global_balance_quote_user.to_account_info(), accounts.token_program.
            to_account_info(), close_bot_param.global_quote_bump, revenue)?;
    }
    emit!(CloseEvent{
        account_id: accounts.grid_bot.user,
        bot_id: accounts.grid_bot.bot_id.to_string(),
        user_state_id: close_bot_param.user_state_id.to_string(),
        refund: 0,
    });
    Ok(())
}
```

Listing 2.10: src/process/grid/close.rs

Impact The actual `revenue` extracted by the user does not match the expected amount.

Suggestion Replace `total_quote_amount` with `revenue`.

2.2.3 Incorrect logic implemented in function `round_up()`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the function `round_up()`, the condition `scaled > value` should be changed to `scaled < value`, ensuring that rounding up occurs when the scaled value is smaller than the original value after scaling.

```rust
pub fn round_up(value: BigDecimal, scale: i64) -> BigDecimal {
 let scaled = value.with_scale(scale); // This ensures the decimal is scaled correctly
 if scaled > value {
     // If scaling down truncates the value, we need to round up
     scaled + BigDecimal::new(1.into(), scale)
 } else {
     // If scaling down does not truncate (i.e., it's already an exact scale decimal), return as
         is
     scaled
 }
}
```

Listing 2.11: src/common/big_decimal.rs

Impact Incorrect implementation of `round_up()`.

Suggestion Replace `scaled > value` with `scaled < value`.

2.2.4 Incorrect token account and corresponding authority used in function `close_bot()`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Users can close their own `grid_bot` through the function `close_bot()`. In the function `internal_close_bot()`, the procedure involves subtracting the corresponding revenue from the total_base_amount or total_quote_amount recorded in the grid_bot, and then withdrawing the user's principal and profit from the protocol's token accounts. However, when the revenue_token is base_token, there are errors with the authority, protocol token account, and user token account. Specifically, the account associated with base_token should be used instead of the one associated with quote_token.

```rust
pub fn close_bot(ctx: Context<GridBotClose>, close_bot_param: CloseBotParam) -> Result<()> {
    check_context(&ctx)?;
    return GridProcess::close_bot(ctx.accounts, close_bot_param);
}
```

Listing 2.12: lib.rs

```rust
pub fn close_bot(accounts: &mut GridBotClose, close_bot_param: CloseBotParam) -> Result<()> {
   require!(!accounts.grid_bot.closed, Errors::InvalidBotStatus);
   require!(accounts.grid_bot.user == *accounts.user.key, Errors::InvalidUser);

   return GridProcess::internal_close_bot(accounts, close_bot_param);
}
```

Listing 2.13: src/process/grid/close.rs

```rust
pub fn internal_close_bot(accounts: &mut GridBotClose, close_bot_param: CloseBotParam) ->
     Result<()> {
   // sign closed
   accounts.grid_bot.closed = true;
   // harvest revenue, must fist execute, will split revenue from bot's asset
   let (revenue_token, revenue) = GridProcess::internal_harvest_revenue(accounts.grid_bot.
       as_mut(), accounts.pair.as_ref());

   // withdraw token
   GridProcess::internal_withdraw_asset(&accounts.grid_bot_state.key(), &accounts.base_mint.key
       (), accounts.global_balance_base.to_account_info(), accounts.user_base_token_account.
       to_account_info(),
   accounts.global_balance_base_user.to_account_info(), accounts.token_program.to_account_info
       (), close_bot_param.global_base_bump, accounts.grid_bot.total_base_amount.clone())?;
   GridProcess::internal_withdraw_asset(&accounts.grid_bot_state.key(), &accounts.quote_mint.
       key(), accounts.global_balance_quote.to_account_info(), accounts.
       user_quote_token_account.to_account_info(),
   accounts.global_balance_quote_user.to_account_info(), accounts.token_program.to_account_info
       (), close_bot_param.global_quote_bump, accounts.grid_bot.total_quote_amount.clone())?;
   if revenue_token == accounts.base_mint.key() {
       GridProcess::internal_withdraw_asset(&accounts.grid_bot_state.key(), &accounts.base_mint
           .key(), accounts.global_balance_quote.to_account_info(), accounts.
           user_quote_token_account.to_account_info(),
       accounts.global_balance_quote_user.to_account_info(), accounts.token_program.
           to_account_info(), close_bot_param.global_quote_bump, accounts.grid_bot.
           total_quote_amount.clone())?;
   } else {
       GridProcess::internal_withdraw_asset(&accounts.grid_bot_state.key(), &accounts.
           quote_mint.key(), accounts.global_balance_quote.to_account_info(), accounts.
           user_quote_token_account.to_account_info(),
        accounts.global_balance_quote_user.to_account_info(), accounts.token_program.
            to_account_info(), close_bot_param.global_quote_bump, revenue)?;
    }
    emit!(CloseEvent{
        account_id: accounts.grid_bot.user,
        bot_id: accounts.grid_bot.bot_id.to_string(),
        user_state_id: close_bot_param.user_state_id.to_string(),
        refund: 0,
    });
    Ok(())
}
```

Listing 2.14: src/process/grid/close.rs

Impact When the `revenue_token` is `base_token`, the function `close_bot()` fails to execute properly.

Suggestion Revise the logic to ensure that when the `revenue_token` is `base_token`, all related accounts are correctly associated with the `base_token`.

2.3 Additional Recommendation

2.3.1 Lack of check for the length of `maker_users`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the function `set_maker_user()`, it should be ensured that the length of `maker_users.users` is less than the reserved space of 100.

```
pub fn set_maker_user(accounts: &mut MakerUserSet, maker_user: Pubkey, enable: bool) -> Result
    <()> {
    if enable {
        accounts.maker_users.users.push(maker_user);
    } else {
        for (index, user) in accounts.maker_users.users.iter().enumerate() {
            if user.clone() == maker_user {
                accounts.maker_users.users.remove(index);
                break
            }
        }
    }
    Ok(())
}
}
```

Listing 2.15: src/process/admin/maker_user.rs

Suggestion Add length check in the function `set_maker_user()`.

2.3.2 Redundant status configured in `grid_bot`

Status Fixed in `Version 2`

Introduced by Version 1

Description When the user creates a grid_bot, the parameters slippage and entry_price passed in are not actually used. Additionally, the grid_bot's active status would not change even when it is closed, which is redundant.

```rust
pub fn create_bot(accounts: &mut GridBotCreate, name: String, slippage: u16, grid_type: u8,
grid_rate: u16, grid_offset: u64, first_base_amount: u64, first_quote_amount: u64,
last_base_amount: u64, last_quote_amount: u64, fill_base_or_quote: bool, valid_until_time: u64,
entry_price: u64, global_balance_base_user_bump: u8, global_balance_quote_user_bump: u8) -> Result<()> {
require!(!accounts.grid_bot.is_initialized, Errors::Initialized);
require!(name.as_bytes().len() <= MAX_NAME_LENGTH, Errors::InvalidName);
msg!("c addr:{},id:{}", accounts.grid_bot.key(), accounts.grid_bot.bot_id.to_string());


require!(valid_until_time > accounts.clock.unix_timestamp as u64 * 1000, Errors::InvalidUntilTime);
require!(accounts.pair.base_token == accounts.base_mint.key() && accounts.pair.quote_token == accounts.quote_mint.key(), Errors::InvalidPair);
require!(accounts.grid_bot_state.status == GridStatus::Running, Errors::PauseOrShutdown);
require!(accounts.grid_bot.grid_sell_count + accounts.grid_bot.grid_buy_count <= MAX_GRID_COUNT, Errors::MoreThanMaxGridCount);
let (base_amount_sell, quote_amount_buy) = GridProcess::internal_calculate_bot_assets(first_quote_amount as u128, last_base_amount as u128, accounts.grid_bot.grid_sell_count as u128, accounts.grid_bot.grid_buy_count as u128,
grid_type, grid_rate, grid_offset as u128, fill_base_or_quote.clone());
// transfer asset
GridProcess::internal_transfer_asset_to_global_base_balance(accounts, global_balance_base_user_bump, base_amount_sell)?;
GridProcess::internal_transfer_asset_to_global_quote_balance(accounts, global_balance_quote_user_bump, quote_amount_buy)?;
// check balance
GridProcess::internal_check_bot_amount(accounts, accounts.grid_bot.grid_sell_count, accounts.grid_bot.grid_buy_count, first_base_amount as u128, first_quote_amount as u128,
last_base_amount as u128, last_quote_amount as u128, base_amount_sell as u128, quote_amount_buy as u128)?;


// create bot
let pair_key = AdminProcess::internal_get_pair_key(accounts.base_mint.key(), accounts.quote_mint.key());


accounts.grid_bot.is_initialized = true;
accounts.grid_bot.name = name;
accounts.grid_bot.active = true;
accounts.grid_bot.user = *accounts.user.key;
accounts.grid_bot.closed = false;
accounts.grid_bot.pair_id = pair_key;
accounts.grid_bot.grid_type = grid_type;
accounts.grid_bot.grid_rate = grid_rate;
accounts.grid_bot.grid_offset = grid_offset;
accounts.grid_bot.first_base_amount = first_base_amount;
accounts.grid_bot.first_quote_amount = first_quote_amount;
accounts.grid_bot.last_base_amount = last_base_amount;
accounts.grid_bot.last_quote_amount = last_quote_amount;
accounts.grid_bot.fill_base_or_quote = fill_base_or_quote;
accounts.grid_bot.valid_until_time = valid_until_time;
accounts.grid_bot.total_base_amount = base_amount_sell;
accounts.grid_bot.total_quote_amount = quote_amount_buy;
accounts.grid_bot.revenue = 0;
accounts.grid_bot.total_revenue = 0;


let user_state_id = accounts.user_state.next_user_bot_id;
accounts.user_state.next_user_bot_id += 1;
// create event
let pair: &Pair = &accounts.pair;
last_base_amountlet grid_bot: &GridBot = &accounts.grid_bot;
emit!(CreateEvent{
account_id: *accounts.user.key,
bot_id: accounts.grid_bot.bot_id.to_string(),
user_state_id: user_state_id.to_string(),
base_price: "".to_string(),
quote_price: "".to_string(),
base_expo: "".to_string(),
quote_expo: "".to_string(),
slippage: slippage,
entry_price: entry_price,
pair: pair_to_output(pair),
grid_bot: grid_bot_to_output(grid_bot),
});
Ok(())
}
```

Listing 2.16: src/process/grid/create.rs

Suggestion Remove the redundant status and parameters.

Feedback from the project The parameter `entry_price` will be used for off-chain recording.

2.3.3 Lack of check for the sum of `grid_sell_count` and `grid_buy_count`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the function `create_bot_account()`, it should be ensured that `grid_sell_count` + `grid_buy_count` > 0.

```
pub fn create_bot_account(accounts: &mut GridBotAccountCreate, grid_sell_count: u16,
    grid_buy_count: u16) -> Result<()> {
  require!(!accounts.grid_bot.is_initialized, Errors::Initialized);
  let empty = GridProcess::grid_bot_is_empty(accounts.grid_bot.as_ref());
  accounts.grid_bot.bot_id = accounts.grid_bot_state.next_bot_id;
  accounts.grid_bot.grid_sell_count = grid_sell_count;
```

```
    accounts.grid_bot.grid_buy_count = grid_buy_count;
    if empty {
        accounts.grid_bot_state.next_bot_id += 1;
    }
    msg!("ca addr:{},id:{}", accounts.grid_bot.key(), accounts.grid_bot.bot_id.to_string());
    Ok(())
}
```

Listing 2.17: src/process/grid.rs/create.rs

Suggestion Add relevant checks in the function `create_bot_account()`.

2.3.4 Saving bumps when initializing PDA accounts for later use

Status Confirmed

Introduced by Version 1

Description In functions that involve transfer operations, the user always needs to pass in the `bumps` for the relevant `PDA`. Specifically, for instance, in the `close_bot()` function, the user needs to pass in the `global_base_bump` and `global_quote_bump` for the purpose of cross-contract transfers. However, these `bumps` could entirely be stored as data when creating the corresponding `PDA`, and then read and used directly when needed.

```
pub fn close_bot(accounts: &mut GridBotClose, close_bot_param: CloseBotParam) -> Result<()> {
    require!(!accounts.grid_bot.closed, Errors::InvalidBotStatus);
    require!(accounts.grid_bot.user == *accounts.user.key, Errors::InvalidUser);

    return GridProcess::internal_close_bot(accounts, close_bot_param);
}

pub fn internal_close_bot(accounts: &mut GridBotClose, close_bot_param: CloseBotParam) ->
     Result<()> {
    // sign closed
    accounts.grid_bot.closed = true;
    // harvest revenue, must fist execute, will split revenue from bot's asset
    let (revenue_token, revenue) = GridProcess::internal_harvest_revenue(accounts.grid_bot.
        as_mut(), accounts.pair.as_ref());

    // withdraw token
    GridProcess::internal_withdraw_asset(&accounts.grid_bot_state.key(), &accounts.base_mint.key
        (), accounts.global_balance_base.to_account_info(), accounts.user_base_token_account.
        to_account_info(),
    accounts.global_balance_base_user.to_account_info(), accounts.token_program.to_account_info
        (), close_bot_param.global_base_bump, accounts.grid_bot.total_base_amount.clone())?;
    GridProcess::internal_withdraw_asset(&accounts.grid_bot_state.key(), &accounts.quote_mint.
        key(), accounts.global_balance_quote.to_account_info(), accounts.
        user_quote_token_account.to_account_info(),
    accounts.global_balance_quote_user.to_account_info(), accounts.token_program.to_account_info
        (), close_bot_param.global_quote_bump, accounts.grid_bot.total_quote_amount.clone())?;
    if revenue_token == accounts.base_mint.key() {
        GridProcess::internal_withdraw_asset(&accounts.grid_bot_state.key(), &accounts.base_mint
            .key(), accounts.global_balance_quote.to_account_info(), accounts.
            user_quote_token_account.to_account_info(),
        accounts.global_balance_quote_user.to_account_info(), accounts.token_program.
            to_account_info(), close_bot_param.global_quote_bump, accounts.grid_bot.
            total_quote_amount.clone())?;
    } else {
        GridProcess::internal_withdraw_asset(&accounts.grid_bot_state.key(), &accounts.
            quote_mint.key(), accounts.global_balance_quote.to_account_info(), accounts.
            user_quote_token_account.to_account_info(),
        accounts.global_balance_quote_user.to_account_info(), accounts.token_program.
            to_account_info(), close_bot_param.global_quote_bump, revenue)?;
    }
    emit!(CloseEvent{
        account_id: accounts.grid_bot.user,
        bot_id: accounts.grid_bot.bot_id.to_string(),
        user_state_id: close_bot_param.user_state_id.to_string(),
        refund: 0,
    });
    Ok(())
}
```

Listing 2.18: src/process/grid/close.rs

Suggestion Store bumps during the initiation for later use.

2.3.5 Lack of access control during the initialization

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `initialize()` is used to initialize the contract account, but it lacks access control. Specifically, once the contract is deployed on the blockchain, any account can invoke the function `initialize()`.

```
  pub fn initialize(ctx: Context<Initialize>) -> Result<()> {
    check_context(&ctx)?;
    return AdminProcess::initialize(ctx.accounts);
}
```

Listing 2.19: src/lib.rs

Suggestion Add access control to ensure that only a specified account can invoke the function `initialize()`.

2.3.6 Redundant accounts required in function `register_pair()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the function `register_pair()`, the struct `RegisterPair` contains many redundant accounts, such as `global_balance_base_user`.

```
  pub fn register_pair(ctx: Context<RegisterPair>, base_min_deposit: u64, quote_min_deposit: u64
      ) -> Result<()> {
    check_context(&ctx)?;
    check_owner(&ctx.accounts.grid_bot_state.owner_id, ctx.accounts.user.key)?;
    return AdminProcess::register_pair(ctx.accounts, base_min_deposit, quote_min_deposit);
}
```

Listing 2.20: src/lib.rs

```
pub struct RegisterPair<'info> {
    pub grid_bot_state: Box<Account<'info, GridBotState>>,
    #[account(init_if_needed, payer = user, space = 8 + 64, seeds = [grid_bot_state.key().as_ref
        (), base_mint.key().as_ref(), quote_mint.key().as_ref()], bump)]
    pub pair: Box<Account<'info, Pair>>,


    pub base_mint: Box<InterfaceAccount<'info, Mint>>,
    /// CHECK:global_balance_base_user
    #[account(seeds = [PDA_SEED_GLOBAL_BALANCE_USER.as_bytes(), grid_bot_state.key().as_ref(),
        base_mint.key().as_ref()], bump)]
    pub global_balance_base_user: UncheckedAccount<'info>,
    #[account(associated_token::mint = base_mint, associated_token::authority =
        global_balance_base_user)]
    pub global_balance_base: Box<InterfaceAccount<'info, TokenAccount>>,


    pub quote_mint: Box<InterfaceAccount<'info, Mint>>,
    /// CHECK:global_balance_base_user
    #[account(seeds = [PDA_SEED_GLOBAL_BALANCE_USER.as_bytes(), grid_bot_state.key().as_ref(),
        quote_mint.key().as_ref()], bump)]
    pub global_balance_quote_user: UncheckedAccount<'info>,
    #[account(associated_token::mint = quote_mint, associated_token::authority =
        global_balance_quote_user)]
    pub global_balance_quote: Box<InterfaceAccount<'info, TokenAccount>>,


    #[account(seeds = [PDA_SEED_PROTOCOL_BALANCE_USER.as_bytes(), grid_bot_state.key().as_ref(),
         base_mint.key().as_ref()], bump)]
    pub protocol_balance_base_record: Box<Account<'info, DataRecord>>,


    #[account(seeds = [PDA_SEED_PROTOCOL_BALANCE_USER.as_bytes(), grid_bot_state.key().as_ref(),
         quote_mint.key().as_ref()], bump)]
    pub protocol_balance_quote_record: Box<Account<'info, DataRecord>>,


    #[account(mut, seeds = [PDA_SEED_DEPOSIT_LIMIT.as_bytes(), grid_bot_state.key().as_ref(),
        base_mint.key().as_ref()], bump)]
    pub deposit_limit_base: Box<Account<'info, DataRecord>>,
    #[account(mut, seeds = [PDA_SEED_DEPOSIT_LIMIT.as_bytes(), grid_bot_state.key().as_ref(),
        quote_mint.key().as_ref()], bump)]
    pub deposit_limit_quote: Box<Account<'info, DataRecord>>,


    pub token_program: Interface<'info, TokenInterface>,
    pub associated_token_program: Program<'info, AssociatedToken>,
    #[account(mut)]
    pub user: Signer<'info>,
    pub system_program: Program<'info, System>,
}
```

Listing 2.21: src/instruction/admin/register_pair.rs

Suggestion Remove the redundant accounts.

2.3.7 Lack of check in function `set_owner()`

Status Confirmed

Introduced by `Version 1`

Description The `set_owner()` function allows the privileged `owner` to transfer ownership. However, it does not validate whether the new `owner` is the same as the previous one. Additionally, since transferring ownership is a sensitive operation, it would be better to make the process two steps - firstly setting a pending `owner`, and then having the pending `owner` accept and become the new `owner`. This would add an extra security check during the transfer of ownership.

```
pub fn set_owner(ctx: Context<SetOwner>) -> Result<()> {
  check_context(&ctx)?;
  check_owner(&ctx.accounts.grid_bot_state.owner_id, ctx.accounts.user.key)?;
  return AdminProcess::set_owner(ctx.accounts);
}
```

Listing 2.22: src/lib.rs

```
/// Change owner. Only can be called by owner.
pub fn set_owner(accounts: &mut SetOwner) -> Result<(), Error> {
   accounts.grid_bot_state.owner_id = *accounts.new_owner_id.key;
   Ok(())
}
```

Listing 2.23: src/process/admin/owner.rs

Suggestion Revise the logic accordingly.

Feedback from the project The ownership will be transferred to a multi-sig wallet before going live, and the multi-sig itself will execute the owner replacement logic, which has already undergone multiple verifications.

2.3.8 Lack of duplication check in function `set_maker_user()`

Status Confirmed

Introduced by `Version 1`

Description The protocol owner can use the function `set_maker_user()` to add any account to the whitelist. Only accounts in the whitelist can interact as takers and with orders in the `grid_bot`. The protocol uses a `vector` to record the whitelist, and there is no check to determine whether an account is already on the whitelist when adding an account. This can potentially lead to the same account being added multiple times.

```rust
pub fn set_maker_user(ctx: Context<MakerUserSet>, maker_user: Pubkey, enable: bool) -> Result
    <()> {
    check_context(&ctx)?;
    check_owner(&ctx.accounts.grid_bot_state.owner_id, ctx.accounts.user.key)?;
    return AdminProcess::set_maker_user(ctx.accounts, maker_user, enable);
}
```

Listing 2.24: src/lib.rs

```rust
pub fn set_maker_user(accounts: &mut MakerUserSet, maker_user: Pubkey, enable: bool) -> Result
    <()> {
    if enable {
        accounts.maker_users.users.push(maker_user);
    } else {
        for (index, user) in accounts.maker_users.users.iter().enumerate() {
            if user.clone() == maker_user {
                accounts.maker_users.users.remove(index);
                break
            }
        }
    }
    Ok(())
}
```

Listing 2.25: src/process/admin/maker_user.rs

Suggestion Revise the logic to ensure that the account being added is not already in the whitelist.

Feedback from the project It's designed as purpose, as there this a function implemented for deleting it. No need to introduce excessive unnecessary logic in the contract.

2.4 Note

2.4.1 Potential centralization risks

Introduced by `Version 1`

Description The protocol includes several privileged functions, such as `set_protocol_fee_rate()`, and `set_maker_user()`. If the `owner`'s private key is lost or maliciously exploited, it could potentially cause losses to users.

Feedback from the project There is a maximum value check (20%), and it only applies to the user's profit of 20%, which will not affect the principal.

BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS